

## **REMARKS**

The Examiner rejected claims 1-18 under 35 U.S.C. §102 as anticipated by Foody.

Before noting why the claims distinguish over Foody, first the claims will be explained. Using claim 1 as exemplary, the following is noted. Claim 1 first recites objects comprising software components. Software components are parts of software. As further recited by claim 1, the software components have dynamically linkable named inputs and outputs stored on a memory of the computer system. As shown in Fig. 6 of Applicants' drawings, three of such software components are shown with inputs and outputs. Note in Fig. 6 that the output of the software component "number generator component" is linked (that is connected) to the software component "printer component". Note that the name of the output and input which are linked is "numbers". Note that for the other software component "multiplier component" that the input and output are respectively named "mult-in" and "mult-out".

As next recited in claim 1, the event communication framework provides automated pattern-based fully distributable events such that when a new software component is loaded into the computer system also having dynamically linkable named inputs and outputs, the new software component inputs and outputs are automatically linked to the inputs and outputs of the *same* name as said stored software components. This can be seen in Fig. 6 where, for example, if "the software component "number generator component" was the new software component loaded into the computer, then its output is linked to the stored software component "printer component" having an input with the same name (that is "numbers") but is not linked to the other input or output of the other software

component stored ("multiplier component") having different input and output names. This is also shown in another example in Fig. 7 wherein if the "printer component" and "multiplier component" are the previously stored software components, and "number generator component" is the newly loaded software component, then the inputs and outputs of the same name are automatically connected.

Finally, claim 1 recites that the software components are combined substantially without changing code and without writing adapters. This means that there is no need to *change* code – defined as modifying, adding, or subtracting code - and no need to write adapters.

In the Foody reference cited by the Examiner, claims 1, 2, 3, and 4 of Foody all require changing of code (that is modifying, adding, or subtracting code) and/or writing adapters in order to combine the objects. In claim 1 as recited at line 17 a "proxy means for creating a proxy object that is an object of the first object system and that corresponds to the object of the second object model" is an adapter or changed code which must be written. The proxy object is an adapter and/or code between the first object system and the second object model which must be created. This thus contradicts claim 1 of the present application. Similarly in dependent claim 2 of Foody he states that the "proxy means comprises means for creating a plurality of proxy objects, each proxy being an object of the first object system and corresponding with an object of one of the plurality of object models." Here again this proxy means is an adapter and/or changed code which needs to be created. This is directly contrary to claim 1. Similarly in dependent claim 3 Foody teaches that "the device includes means for adding or removing support for object systems and object models, said means for adding or removing support adding or removing support without compiling the device". This again amounts to a changing of code

and/or writing of adapters. The phrase “without compiling the device” does not mean that code is not changed or that adapters are not provided since as previously explained proxy objects must be created which are adapters or changed code of the overall system (a code addition for the proxy objects).

Finally, in claim 4 Foody teaches “means for sub-classing from object classes of a plurality of object systems”. This sub-classing from object classes is another teaching of changing code (that is adding code).

Claim 1 further distinguishes because nowhere in Foody is there a combination of the objects without changing code and without writing adapters by automatically combining the inputs and outputs of a newly loaded software component with the previously stored software components by automatically linking the inputs and outputs *of the same name*. This was previously explained in relation to Figs. 6 and 7 of Applicants drawings. Nowhere in the drawing Figures 1-15 of Foody is there any showing of the automatic connection of the same name inputs and outputs, such as shown in the exemplary embodiment of Figures 6 and 7 of Applicants specification, particularly without changing code (that is addition or subtraction of code) or without adding adapters. Foody needs proxy objects.

Dependent claims 2-4 distinguish at least for the reasons claim 1 distinguishes and also by reciting additional features not suggested.

The remaining independent claims 5, 9, 14 and 18 distinguish in a similar fashion as discussed above for claim 1. The same is true of the respective dependent claims.

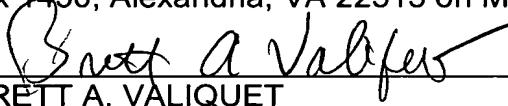
Respectfully submitted,

  
\_\_\_\_\_  
(Reg). #27,841

Brett A. Valiquet  
Schiff Hardin LLP  
Patent Department  
6600 Sears Tower  
Chicago, Illinois 60606  
Telephone: 312-258-5786  
Attorneys for Applicants  
**CUSTOMER NO. 26574**

**CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313 on May 3, 2005.

  
\_\_\_\_\_  
BRETT A. VALIQUET

CHI\4251006.1